DRAFT:
~~Started~~
6/19/67

Identification

Hard-Core Supervisor Callout Mechanism

R. L. Rappaport

Purpose

At system initialization time, the intersegment references made between hardcore supervisor modules are prelinked. References from hardcore modules to segments in other rings cannot be linked at initialization time since the segments to which the references are directed, are not available at this time. Linking at execution time cannot occur in the normal fashion (i.e. by an ft2 fault in the hardcore ring) since the linker is in the administrative ring. The mechanism described here allows the linking to be accomplished at execution time by arranging for the linkage faults to occur in the ring to which the call out is directed.

Introduction

In order to simplify linking of calls out from the hardcore ring all the calls are directed through one procedure. That is, only one hardcore module ever calls out to any segments not in the hardcore ring. This procedure is known as hcs_out and it has an entry point for each distinct procedure which can be called from a hardcore ring module. Calls out are then handled in the following way.

Hcs_out has a distinct copy of its linkage section in each protection ring. Hcs_out is always entered through the copy of its linkage section in the hardcore ring. Upon entry, hcs_out determines to which protection ring the call is directed, and then reloads base pair 1b←1b to point to the copy of the linkage

section that resides in that ring. At this point, hcs_out merely calls, that is, transfers through its new linkage section to the desired procedure and the linkage fault will occur in this new linkage section residing in the entered ring.

Let us present an example to clarify the above. Figure 1 illustrates the problem.

Suppose segment <a> in the hardcore ring desires to call segment <b> in the administrative ring. The code in segment <a> would be:

```
call hcs_out $ b;
```

At system initialization time <a.link> is linked directly to <hcs_out.link>, the linkage section of hcs_out in the hardcore ring. The flow of control is as follows: <a> transfers through its linkage section to the hardcore linkage section of hcs_out, which in turn transfers directly to the entry hcs_out $ b. Hcs_out knows (or can calculate) that <b> resides in the administrative ring. Therefore hcs_out executes an eaplp for <hcs_out1.link> and then merely calls <b>, that is, transfers through <hcs_out1.link> to <b.link>. If this is the first time that <b> has been called from hcs_out, then a linkage fault will occur in <hcs_out1.link>.

Discussion

Each process has its own distinct version of hcs_out. Segment <hcs_out> consists of a number of entries and an array of pointers to its various linkage sections, one per ring. Initially this array of linkage pointers is filled with "software simulated" fault word pairs (see Section BB.5.03).

Reference to one of these fault word pairs will generate software simulated fault number 4. This fault is serviced in the hardcore ring by creating a copy of <hcs_out.link> in the appropriate ring and filling the faulting word pair with a pointer to this new linkage section. In this way the various linkage sections can be created dynamically only as they are needed.

Hcs_out is used by all procedures, pre-linked at system initialization time, which have reason to call procedures in other rings. For example, the fault interceptor, which resides in all rings, is pre-linked at initialization time. All references by the fault interceptor to procedures outside the hardcore ring are therefore directed through hcs_out, although if the fault interceptor is in ring n calling a ring n procedure, one would not ordinarily consider this a call out from the hardcore ring.

Let us go through an example to illustrate this. Suppose the fault interceptor operating in ring 3 wants to call subroutine signal. The flow of control is as follows (see figure 2):

1. The fault interceptor (<fim> in the diagram) executes a call that looks like:

   call hcs_out$signal (arglist)

   where arglist is the list of arguments expected by the called procedure. This results in a transfer through its linkage section (<fim.link>) to the hardcore linkage section of hcs_out, with which it was pre-linked at initialization time. That is, <fim.link> transfers to <hcs_out0.link>. This implies that both <fim.link> and <hcs_out0.link> reside in each ring.

2. In <hcs_out0.link> is a standard entry which sets lb←lp to point to this linkage section and transfers to the procedure. That is, even though this is being executed in ring 3, entry to hcs_out is still made through its hardcore linkage section.

3. Hcs_out then calls subroutine getring to find out what ring it is currently executing in. That is, it determines to which linkage section it should switch.

4. Hcs_out then reloads lb←lp to point to the copy of its linkage that exists in the ring currently active. [Reloading the linkage pointer will cause generation of software simulated fault number 4 if this is the first time hcs_out has executed in this ring. This potential fault is irrelevant to the operation of hcs_out. Its handling makes it completely transparent. The handling of this fault is described below.]

5. Finally, hcs_out merely calls signal. That is, it transfers through its linkage section to signal possibly causing a linkage fault to occur. Entry to signal is made through its linkage section in that ring.

[Note: Segments which reside in more than one ring must have resident linkage segments in these rings. If the segment expects to dynamically link to other segments, as in the case of signal, then distinct copies of the linkage section must be in each distinct ring. If the segment is pre-linked and if its pre-linked linkage section is read only, as in the case of the fault interceptor, then the same linkage can reside in each ring.]

In this way one can see how the fault interceptor, which is linked to hcs_out at system initialization time, is able to reach signal dynamically at execution time.

Software Simulated Fault Four

This fault is signaled in the hardcore ring. The fault is generated by the following;

```
        eaplp   X,*
          .
          .
          .
```

X:

(777777)8	its
4	*

Any eaplp instruction with an operand like this will generate the fault.
The only legitimate place for such a ~~word pair~~ set of words to exist is in ~~the array of pointers in~~ segment <hcs_out>. The handler for this fault (master mode since it must write in read only <hcs_out>) checks to see if ~~the faulting word pair~~ they do reside in this segment. If ~~they~~ they do the appropriate copy of <hcs_out.link> is made in the appropriate ring and the word pair at X is replaced by an ITS pointer. If the faulting ~~pair word~~ instruction is not in this segment an error is signaled.

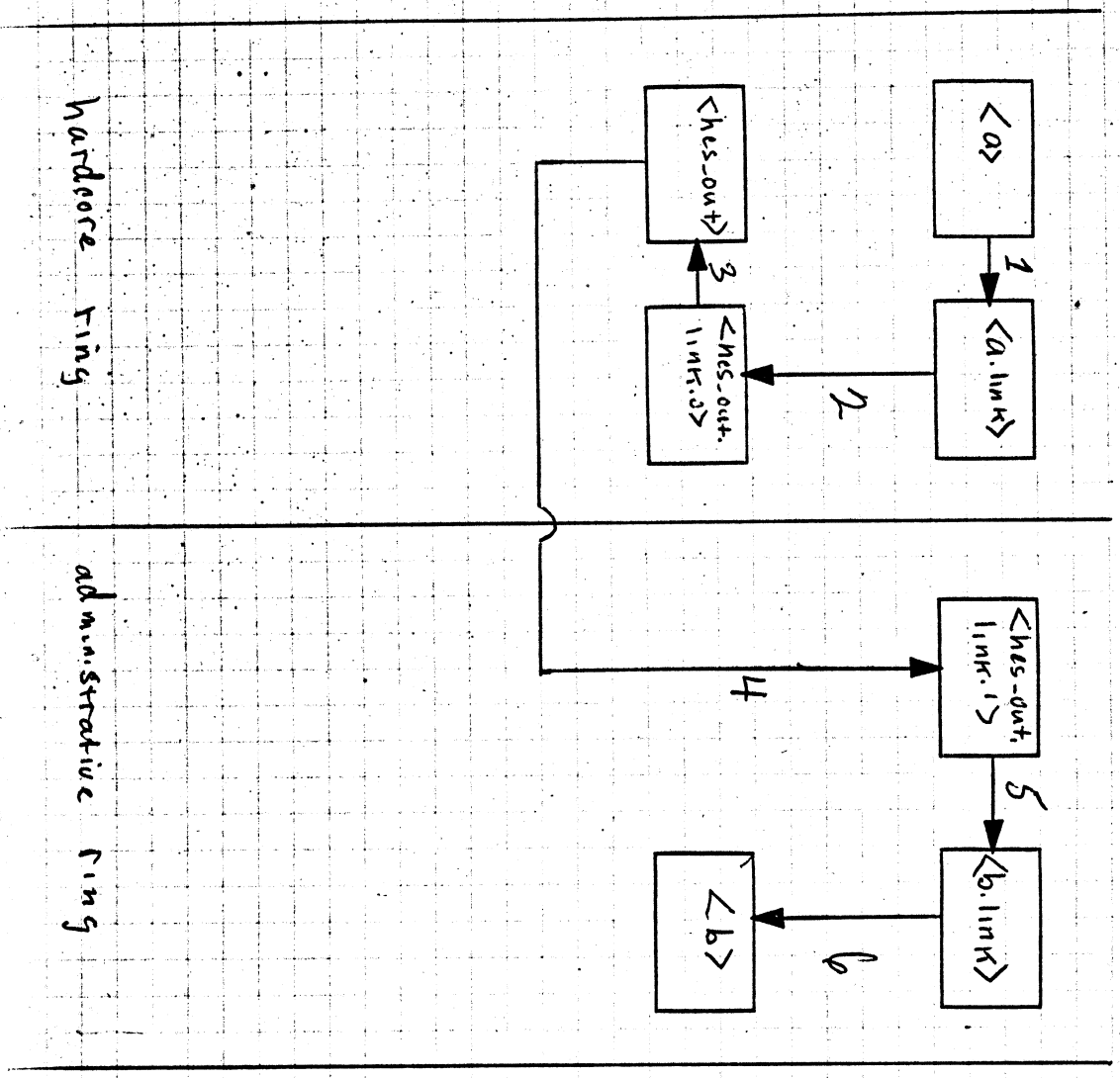

Figure 1. Numerals indicate sequential steps in the flow of control.

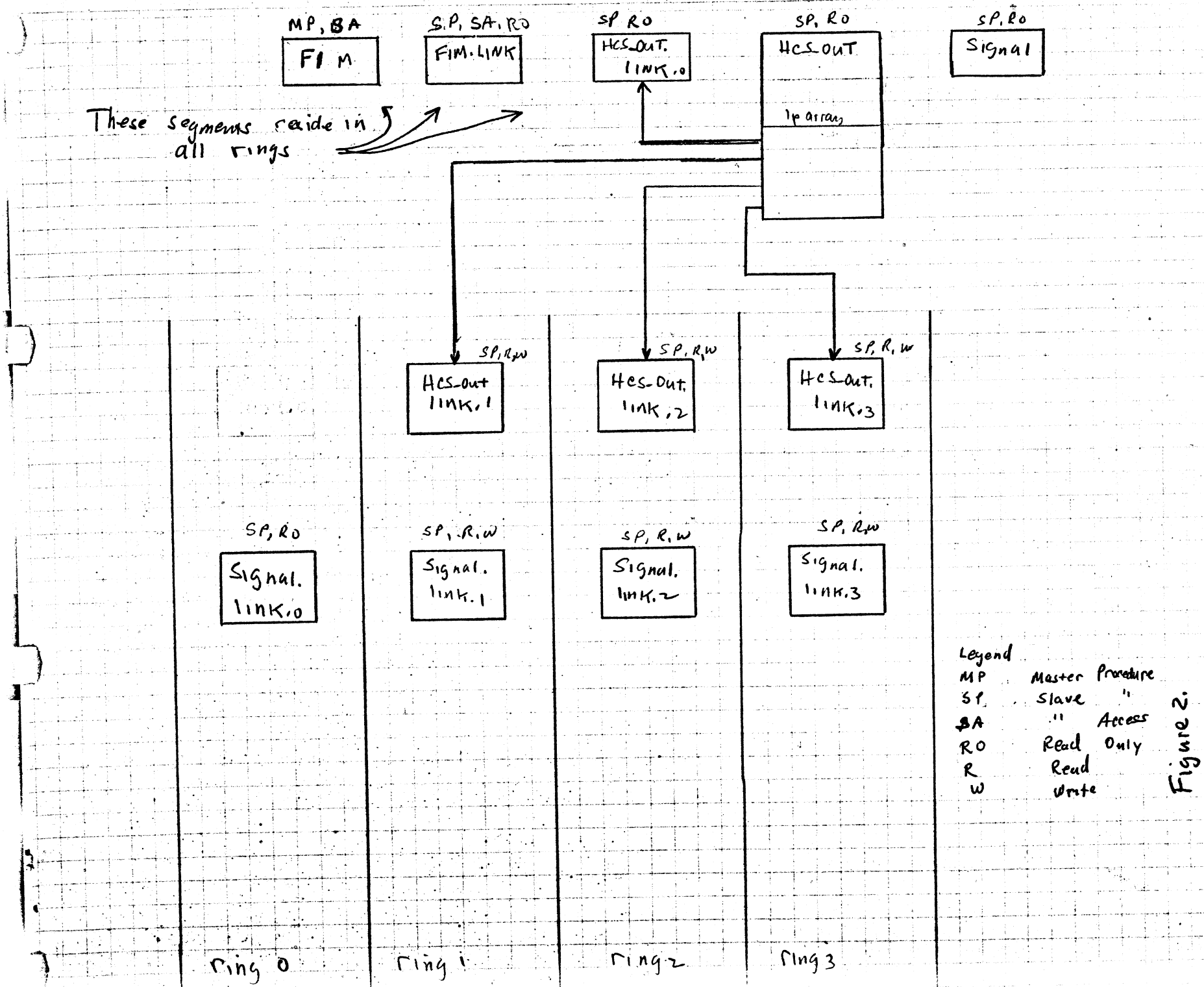
MP, BA
FIM
S.P, SA, RO
FIM.LINK
SP RO
HCS_OUT. LINK.0
SP, RO
HCS_OUT
1p array
SP, RO
Signal
These segments reside in all rings
SP, R,W
HCS_out LINK.1
SP, R,W
Hcs_Out. LINK,2
SP, R, W
HCS_out. LINK.3
SP, RO
Signal. link.0
SP, R.W
Signal. link.1
SP, R, W
Signal. link.2
SP, R,W
Signal. link.3
Ring 0
Ring 1
Ring 2
Ring 3
Legend
MP Master Procedure
SP " Slave "
BA " Access
RO Read Only
R Read
W Write

Figure 2.